

# Network Newton

Aryan Mokhtari<sup>†</sup>, Qing Ling<sup>\*</sup> and Alejandro Ribeiro<sup>†</sup>

<sup>†</sup>Dept. of Electrical and Systems Engineering, University of Pennsylvania

<sup>\*</sup>Dept. of Automation, University of Science and Technology of China

**Abstract**—We consider minimization of a sum of convex objective functions where the components of the objective are available at different nodes of a network and nodes are allowed to only communicate with their neighbors. The use of distributed subgradient or gradient methods is widespread but they often suffer from slow convergence since they rely on first order information, which leads to a large number of local communications between nodes in the network. In this paper we propose the Network Newton (NN) method as a distributed algorithm that incorporates second order information via distributed evaluation of approximations to Newton steps. We also introduce adaptive (A)NN in order to establish exact convergence. Numerical analyses show significant improvement in both convergence time and number of communications for NN relative to existing (first order) alternatives.

## I. INTRODUCTION

Distributed optimization algorithms are used to minimize a global cost function over a set of nodes in situations where the objective function is defined as a sum of a set of local functions. Consider a variable  $\mathbf{x} \in \mathbb{R}^p$  and a connected network containing  $n$  agents each of which has access to a local function  $f_i : \mathbb{R}^p \rightarrow \mathbb{R}$ . The agents cooperate in minimizing the aggregate cost function  $f : \mathbb{R}^p \rightarrow \mathbb{R}$  taking values  $f(\mathbf{x}) := \sum_{i=1}^n f_i(\mathbf{x})$ . I.e., agents cooperate in solving the global optimization problem

$$\mathbf{x}^* := \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{i=1}^n f_i(\mathbf{x}). \quad (1)$$

Problems of this form arise often in, e.g., wireless systems [1], [2], sensor networks [3], [4], and large scale machine learning [5]. There are different algorithms to solve (1) in a distributed manner. The most popular alternatives are decentralized gradient descent (DGD) [6]–[9], distributed implementations of the alternating direction method of multipliers [3], [10]–[12], and decentralized dual averaging (DDA) [13]. A feature common to all of these algorithms is the slow convergence rate in ill-conditioned problems since they operate on first order information only.

This paper considers Network Newton (NN), a method that relies on distributed approximations of Newton steps for the global cost function  $f$  to accelerate convergence of the DGD algorithm. We begin this paper by introducing the idea that DGD solves a penalized version of (1) using gradient descent in lieu of solving the original optimization problem. To accelerate the convergence of gradient descent method for solving the penalty version of (1) we advocate the use of the NN algorithm. This algorithm relies on approximations to the Newton step of the penalized objective function by truncating the Taylor series of the exact Newton step (Section II-A). These approximations to the Newton step can be computed in a distributed manner with a level of locality controlled by the number  $K$  of elements that are retained in the Taylor's series. When we retain  $K$  elements in the series we say that we implement NN- $K$ . We prove that for a fixed penalty coefficient lower and upper bounds on the Hessians of local objective functions  $f_i$  are sufficient to guarantee at least linear convergence of NN- $K$  to the optimal arguments of penalized optimization problem (Theorem 1). Further, We introduce an adaptive version of NN- $K$  (ANN- $K$ ) that uses an increasing penalty coefficient to achieve exact convergence to the optimal solution of (1) (Section II-B). We study the advantages of NN- $K$  relative to DGD, both in terms of number of iterations and communications for convergence for solving a family of quadratic objective problems (Section IV).

## II. PROBLEM FORMULATION AND ALGORITHM DEFINITION

The network that connects the agents is assumed symmetric and specified by the neighborhoods  $\mathcal{N}_i$  that contain the list of nodes than can communicate with  $i$  for  $i = 1, \dots, n$ . DGD is an established distributed method to solve (1) which relies on the introduction of local variables  $\mathbf{x}_i \in \mathbb{R}^p$  and nonnegative weights  $w_{ij} \geq 0$  that are not null if and only if  $j = i$  or if  $j \in \mathcal{N}_i$ . Letting  $t \in \mathbb{N}$  be a discrete time index and  $\alpha$  a given stepsize, DGD is defined by the recursion

$$\mathbf{x}_{i,t+1} = \sum_{j=1}^n w_{ij} \mathbf{x}_{j,t} - \alpha \nabla f_i(\mathbf{x}_{i,t}), \quad i = 1, \dots, n. \quad (2)$$

Since  $w_{ij} = 0$  when  $j \neq i$  and  $j \notin \mathcal{N}_i$ , it follows from (2) that each agent  $i$  updates its estimate  $\mathbf{x}_i$  of the optimal vector  $\mathbf{x}^*$  by performing an average over the estimates  $\mathbf{x}_{j,t}$  of its neighbors  $j \in \mathcal{N}_i$  and its own estimate  $\mathbf{x}_{i,t}$ , and descending through the negative local gradient  $-\nabla f_i(\mathbf{x}_{i,t})$ . Note that weights  $w_{ij}$  that nodes assign to each other form a weight matrix  $\mathbf{W} \in \mathbb{R}^{n \times n}$  that is symmetric and row stochastic. It is also customary to require the rank of  $\mathbf{I} - \mathbf{W}$  to be  $n - 1$  so that  $\operatorname{null}(\mathbf{I} - \mathbf{W}) = \operatorname{span}(\mathbf{1})$ . If the two assumptions  $\mathbf{W}^T = \mathbf{W}$  and  $\operatorname{null}(\mathbf{I} - \mathbf{W}) = \mathbf{1}$  are true, it is possible to show that (2) approaches the solution of (1) in the sense that  $\mathbf{x}_{i,t} \approx \mathbf{x}^*$  for all  $i$  and large  $t$ , [6].

To rewrite (2) define the matrix  $\mathbf{Z} := \mathbf{W} \otimes \mathbf{I} \in \mathbb{R}^{np \times np}$  as the Kronecker product of weight matrix  $\mathbf{W} \in \mathbb{R}^{n \times n}$  and the identity matrix  $\mathbf{I} \in \mathbb{R}^{p \times p}$ . Further, we introduce vectors  $\mathbf{y} := [\mathbf{x}_1; \dots; \mathbf{x}_n] \in \mathbb{R}^{np}$  that concatenates the local vectors  $\mathbf{x}_i$ , and vector  $\mathbf{h}(\mathbf{y}) := [\nabla f_1(\mathbf{x}_1); \dots; \nabla f_n(\mathbf{x}_n)] \in \mathbb{R}^{np}$  which concatenates the gradients of the local functions  $f_i$  taken with respect to the local variable  $\mathbf{x}_i$ . It is then ready to see that (2) is equivalent to

$$\mathbf{y}_{t+1} = \mathbf{Z} \mathbf{y}_t - \alpha \mathbf{h}(\mathbf{y}_t) = \mathbf{y}_t - [(\mathbf{I} - \mathbf{Z}) \mathbf{y}_t + \alpha \mathbf{h}(\mathbf{y}_t)], \quad (3)$$

where in the second equality we added and subtracted  $\mathbf{y}_t$  and regrouped terms. Inspection of (3) reveals that the DGD update formula at step  $t$  is equivalent to a (regular) gradient descent algorithm being used to solve the problem

$$\mathbf{y}^* := \underset{\mathbf{y}}{\operatorname{argmin}} F(\mathbf{y}) := \min_{\mathbf{y}} \frac{1}{2} \mathbf{y}^T (\mathbf{I} - \mathbf{Z}) \mathbf{y} + \alpha \sum_{i=1}^n f_i(\mathbf{x}_i). \quad (4)$$

Observe that it is possible to write the gradient of  $F(\mathbf{y})$  as

$$\mathbf{g}_t := \nabla F(\mathbf{y}_t) = (\mathbf{I} - \mathbf{Z}) \mathbf{y}_t + \alpha \mathbf{h}(\mathbf{y}_t), \quad (5)$$

in order to write (3) as  $\mathbf{y}_{t+1} = \mathbf{y}_t - \mathbf{g}_t$  and conclude that DGD descends along the negative gradient of  $F(\mathbf{y})$  with unit stepsize. The expression in (2) is just a local implementation of (5) where node  $i$  implements the descent  $\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} - \mathbf{g}_{i,t}$  where  $\mathbf{g}_{i,t}$  is the  $i$ th element of the gradient  $\mathbf{g}_t = [\mathbf{g}_{1,t}; \dots; \mathbf{g}_{n,t}]$ . Node  $i$  can compute the local gradient

$$\mathbf{g}_{i,t} = (1 - w_{ii}) \mathbf{x}_{i,t} - \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{x}_{j,t} + \alpha \nabla f_i(\mathbf{x}_{i,t}). \quad (6)$$

Notice that since we know that the null space of  $\mathbf{I} - \mathbf{W}$  is  $\operatorname{null}(\mathbf{I} - \mathbf{W}) = \operatorname{span}(\mathbf{1})$  and that  $\mathbf{Z} = \mathbf{W} \otimes \mathbf{I}$ , we obtain that the span of  $\mathbf{I} - \mathbf{Z}$  is  $\operatorname{null}(\mathbf{I} - \mathbf{Z}) = \operatorname{span}(\mathbf{1} \otimes \mathbf{I})$ . Thus, we have that  $(\mathbf{I} - \mathbf{Z}) \mathbf{y} = \mathbf{0}$  holds if and only if  $\mathbf{x}_1 = \dots = \mathbf{x}_n$ . Since the matrix  $\mathbf{I} - \mathbf{Z}$  is positive semidefinite – because it is stochastic and symmetric –, the same is true of the square root matrix  $(\mathbf{I} - \mathbf{Z})^{1/2}$ . Therefore, we have that the optimization problem

in (1) is equivalent to the optimization problem

$$\tilde{\mathbf{y}}^* := \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{i=1}^n f_i(\mathbf{x}_i), \quad \text{s.t. } (\mathbf{I} - \mathbf{Z})^{1/2} \mathbf{y} = \mathbf{0}. \quad (7)$$

Indeed, for  $\mathbf{y} = [\mathbf{x}_1; \dots; \mathbf{x}_n]$  to be feasible in (7) we must have  $\mathbf{x}_1 = \dots = \mathbf{x}_n$  because  $\operatorname{null}[(\mathbf{I} - \mathbf{Z})^{1/2}] = \operatorname{span}(\mathbf{1} \otimes \mathbf{I})$  as already argued. When restricted to this feasible set the objective  $\sum_{i=1}^n f_i(\mathbf{x}_i)$  of (7) is the same as the objective of (1) from where it follows that a solution  $\tilde{\mathbf{y}}^* = [\tilde{\mathbf{x}}_1^*; \dots; \tilde{\mathbf{x}}_n^*]$  of (7) is such that  $\tilde{\mathbf{x}}_i^* = \tilde{\mathbf{x}}^* = \mathbf{x}^*$  for all  $i$ , i.e.  $\tilde{\mathbf{y}}^* = [\mathbf{x}_1^*; \dots; \mathbf{x}_n^*]$ . The unconstrained minimization in (4) is a penalty version of (7). The penalty function associated with the constraint  $(\mathbf{I} - \mathbf{Z})^{1/2} \mathbf{y} = \mathbf{0}$  is the squared norm  $(1/2)\|(\mathbf{I} - \mathbf{Z})^{1/2} \mathbf{y}\|^2$  and the corresponding penalty coefficient is  $1/\alpha$ . Inasmuch as the penalty coefficient  $1/\alpha$  is sufficiently large, the optimal arguments  $\mathbf{y}^*$  and  $\tilde{\mathbf{y}}^*$  are not too far apart. In this paper we exploit the reinterpretation of (3) as a method to minimize (4) to propose an approximate Newton algorithm that can be implemented in a distributed manner. We explain this algorithm in the following section.

#### A. Network Newton

Instead of solving (4) with a gradient descent algorithm as in DGD, we can solve (4) using Newton's method. To implement Newton's method we need to compute the Hessian  $\mathbf{H}_t := \nabla^2 F(\mathbf{y}_t)$  of  $F$  evaluated at  $\mathbf{y}_t$  so as to determine the Newton step  $\mathbf{d}_t := -\mathbf{H}_t^{-1} \mathbf{g}_t$ . Start by differentiating twice in (4) in order to write  $\mathbf{H}_t$  as

$$\mathbf{H}_t := \nabla^2 F(\mathbf{y}_t) = \mathbf{I} - \mathbf{Z} + \alpha \mathbf{G}_t, \quad (8)$$

where the matrix  $\mathbf{G}_t \in \mathbb{R}^{np \times np}$  is a block diagonal matrix formed by blocks  $\mathbf{G}_{ii,t} \in \mathbb{R}^{p \times p}$  containing the Hessian of the  $i$ th local function,

$$\mathbf{G}_{ii,t} = \nabla^2 f_i(\mathbf{x}_{i,t}). \quad (9)$$

It follows from (8) and (9) that the Hessian  $\mathbf{H}_t$  is block sparse with blocks  $\mathbf{H}_{ij,t} \in \mathbb{R}^{p \times p}$  having the sparsity pattern of  $\mathbf{Z}$ , which is the sparsity pattern of the graph. The diagonal blocks are of the form  $\mathbf{H}_{ii,t} = (1 - w_{ii})\mathbf{I} + \alpha \nabla^2 f_i(\mathbf{x}_{i,t})$  and the off diagonal blocks are not null only when  $j \in \mathcal{N}_i$  in which case  $\mathbf{H}_{ij,t} = w_{ij}\mathbf{I}$ .

While the Hessian  $\mathbf{H}_t$  is sparse, the inverse  $\mathbf{H}_t$  is not. It is the latter that we need to compute the Newton step  $\mathbf{d}_t := -\mathbf{H}_t^{-1} \mathbf{g}_t$ . To overcome this problem we split the diagonal and off-diagonal blocks of  $\mathbf{H}_t$  and rely on a Taylor's expansion of the inverse. To be precise, write  $\mathbf{H}_t = \mathbf{D}_t - \mathbf{B}$  where the matrix  $\mathbf{D}_t$  is defined as

$$\mathbf{D}_t := \alpha \mathbf{G}_t + 2(\mathbf{I} - \operatorname{diag}(\mathbf{Z})) := \alpha \mathbf{G}_t + 2(\mathbf{I} - \mathbf{Z}_d), \quad (10)$$

in the second equality we defined  $\mathbf{Z}_d := \operatorname{diag}(\mathbf{Z})$  for future reference. Since the diagonal weights must be  $w_{ii} < 1$ , the matrix  $\mathbf{I} - \mathbf{Z}_d$  is positive definite. The same is true of the block diagonal matrix  $\mathbf{G}_t$  because the local functions are assumed strongly convex. Therefore, the matrix  $\mathbf{D}_t$  is block diagonal and positive definite. The  $i$ th diagonal block  $\mathbf{D}_{ii,t} \in \mathbb{R}^p$  of  $\mathbf{D}_t$  can be computed and stored by node  $i$  as  $\mathbf{D}_{ii,t} = \alpha \nabla^2 f_i(\mathbf{x}_{i,t}) + 2(1 - w_{ii})\mathbf{I}$ . To have  $\mathbf{H}_t = \mathbf{D}_t - \mathbf{B}$  we must define  $\mathbf{B} := \mathbf{D}_t - \mathbf{H}_t$ . Considering the definitions of  $\mathbf{H}_t$  and  $\mathbf{D}_t$  in (8) and (10), it follows that

$$\mathbf{B} = \mathbf{I} - 2\mathbf{Z}_d + \mathbf{Z}. \quad (11)$$

Observe that  $\mathbf{B}$  is independent of time and depends on the weight matrix  $\mathbf{Z}$  only. As in the case of the Hessian  $\mathbf{H}_t$ , the matrix  $\mathbf{B}$  is block sparse with blocks  $\mathbf{B}_{ij} \in \mathbb{R}^{p \times p}$  having the sparsity pattern of  $\mathbf{Z}$ , which is the sparsity pattern of the graph. Node  $i$  can compute the diagonal blocks  $\mathbf{B}_{ii} = (1 - w_{ii})\mathbf{I}$  and the off diagonal blocks  $\mathbf{B}_{ij} = w_{ij}\mathbf{I}$  using the local information about its own weights only.

Proceed now to factor  $\mathbf{D}_t^{1/2}$  from both sides of the splitting relationship to write  $\mathbf{H}_t = \mathbf{D}_t^{1/2}(\mathbf{I} - \mathbf{D}_t^{-1/2}\mathbf{B}\mathbf{D}_t^{-1/2})\mathbf{D}_t^{1/2}$ . When we consider the Hessian inverse  $\mathbf{H}^{-1}$ , we can use the Taylor series  $(\mathbf{I} - \mathbf{X})^{-1} =$

---

#### Algorithm 1 Network Newton- $K$ method at node $i$

---

**Require:** Initial iterate  $\mathbf{x}_{i,0}$ .

- 1: **for**  $t = 0, 1, 2, \dots$  **do**
  - 2:   Exchange iterates  $\mathbf{x}_{i,t}$  with neighbors  $j \in \mathcal{N}_i$ .
  - 3:   Gradient:  $\mathbf{g}_{i,t} = (1 - w_{ii})\mathbf{x}_{i,t} - \sum_{j \in \mathcal{N}_i} w_{ij}\mathbf{x}_{j,t} + \alpha \nabla f_i(\mathbf{x}_{i,t})$ .
  - 4:   Compute NN-0 descent direction  $\mathbf{d}_{i,t}^{(0)} = -\mathbf{D}_{ii,t}^{-1} \mathbf{g}_{i,t}$
  - 5:   **for**  $k = 0, \dots, K-1$  **do**
  - 6:     Exchange local elements  $\mathbf{d}_{i,t}^{(k)}$  of the NN- $k$  step with neighbors
  - 7:     NN- $(k+1)$  step:  $\mathbf{d}_{i,t}^{(k+1)} = \mathbf{D}_{ii,t}^{-1} \left[ \sum_{j \in \mathcal{N}_i, j=i} \mathbf{B}_{ij} \mathbf{d}_{j,t}^{(k)} - \mathbf{g}_{i,t} \right]$ .
  - 8:   **end for**
  - 9:   Update local iterate:  $\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + \epsilon \mathbf{d}_{i,t}^{(K)}$ .
  - 10: **end for**
- 

$\sum_{j=0}^{\infty} \mathbf{X}^j$  with  $\mathbf{X} = \mathbf{D}_t^{-1/2} \mathbf{B} \mathbf{D}_t^{-1/2}$  to write

$$\mathbf{H}_t^{-1} = \mathbf{D}_t^{-1/2} \sum_{k=0}^{\infty} \left( \mathbf{D}_t^{-1/2} \mathbf{B} \mathbf{D}_t^{-1/2} \right)^k \mathbf{D}_t^{-1/2}. \quad (12)$$

Observe that the sum in (12) converges if the absolute value of all the eigenvalues of matrix  $\mathbf{D}_t^{-1/2} \mathbf{B} \mathbf{D}_t^{-1/2}$  are strictly less than 1. This result is proven in [14]. Network Newton (NN) is defined as a family of algorithms that rely on truncations of the series in (12). The  $K$ th member of this family, NN- $K$  considers the first  $K+1$  terms of the series to define the approximate Hessian inverse

$$\hat{\mathbf{H}}_t^{(K)-1} := \mathbf{D}_t^{-1/2} \sum_{k=0}^K \left( \mathbf{D}_t^{-1/2} \mathbf{B} \mathbf{D}_t^{-1/2} \right)^k \mathbf{D}_t^{-1/2}. \quad (13)$$

NN- $K$  uses the approximate Hessian  $\hat{\mathbf{H}}_t^{(K)-1}$  as a curvature correction matrix that is used in lieu of the exact Hessian inverse  $\mathbf{H}^{-1}$  to estimate the Newton step. I.e., instead of descending along the Newton step  $\mathbf{d}_t := -\mathbf{H}_t^{-1} \mathbf{g}_t$  we descend along the NN- $K$  step  $\mathbf{d}_t^{(K)} := -\hat{\mathbf{H}}_t^{(K)-1} \mathbf{g}_t$ , which we intend as an approximation of  $\mathbf{d}_t$ . Using the explicit expression for  $\hat{\mathbf{H}}_t^{(K)-1}$  in (13) we write the NN- $K$  step as

$$\mathbf{d}_t^{(K)} = -\mathbf{D}_t^{-1/2} \sum_{k=0}^K \left( \mathbf{D}_t^{-1/2} \mathbf{B} \mathbf{D}_t^{-1/2} \right)^k \mathbf{D}_t^{-1/2} \mathbf{g}_t, \quad (14)$$

where, we recall, the vector  $\mathbf{g}_t$  is the gradient of objective function  $F(\mathbf{y})$  defined in (5). The NN- $K$  update formula can then be written as

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \epsilon \mathbf{d}_t^{(K)}. \quad (15)$$

The algorithm defined by recursive application of (15) can be implemented in a distributed manner because the truncated series in (13) has a local structure controlled by the parameter  $K$ . To explain this statement better define the components  $\mathbf{d}_{i,t}^{(K)} \in \mathbb{R}^p$  of the NN- $K$  step  $\mathbf{d}_t^{(K)} = [\mathbf{d}_{1,t}^{(K)}; \dots; \mathbf{d}_{n,t}^{(K)}]$ . A distributed implementation of (15) requires that node  $i$  computes  $\mathbf{d}_{i,t}^{(K)}$  so as to implement the local descent  $\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + \epsilon \mathbf{d}_{i,t}^{(K)}$ . The step components  $\mathbf{d}_{i,t}^{(K)}$  can be computed through local computations. To see that this is true first note that considering the definition of the NN- $K$  descent direction in (14) the sequence of NN descent directions satisfies

$$\mathbf{d}_t^{(k+1)} = \mathbf{D}_t^{-1} \mathbf{B} \mathbf{d}_t^{(k)} - \mathbf{D}_t^{-1} \mathbf{g}_t = \mathbf{D}_t^{-1} (\mathbf{B} \mathbf{d}_t^{(k)} - \mathbf{g}_t). \quad (16)$$

Then observe that since the matrix  $\hat{\mathbf{B}}$  has the sparsity pattern of the graph, this recursion can be decomposed into local components

$$\mathbf{d}_{i,t}^{(k+1)} = \mathbf{D}_{ii,t}^{-1} \left( \sum_{j \in \mathcal{N}_i, j=i} \mathbf{B}_{ij} \mathbf{d}_{j,t}^{(k)} - \mathbf{g}_{i,t} \right), \quad (17)$$

The matrix  $\mathbf{D}_{ii,t} = \alpha \nabla^2 f_i(\mathbf{x}_{i,t}) + 2(1 - w_{ii})\mathbf{I}$  is stored and computed at

**Algorithm 2** Computation of NN- $K$  step at node  $i$ .

---

```

1: function  $\mathbf{x}_i = \text{NN-}K(\alpha, \mathbf{x}_i, \text{tol})$ 
2: while  $\|\mathbf{g}_i\| > \text{tol}$  do
3:   B matrix blocks:  $\mathbf{B}_{ii} = (1 - w_{ii})\mathbf{I}$  and  $\mathbf{B}_{ij} = w_{ij}\mathbf{I}$ 
4:   D matrix block:  $\mathbf{D}_{ii,t} = \alpha \nabla^2 f_i(\mathbf{x}_i) + 2(1 - w_{ii})\mathbf{I}$ 
5:   Exchange iterates  $\mathbf{x}_i$  with neighbors  $j \in \mathcal{N}_i$ .
6:   Gradient:  $\mathbf{g}_i = (1 - w_{ii})\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} w_{ij}\mathbf{x}_j + \alpha \nabla f_i(\mathbf{x}_i)$ .
7:   Compute NN-0 descent direction  $\mathbf{d}_i^{(0)} = -\mathbf{D}_{ii}^{-1}\mathbf{g}_i$ 
8:   for  $k = 0, \dots, K-1$  do
9:     Exchange elements  $\mathbf{d}_i^{(k)}$  of the NN- $k$  step with neighbors
10:    NN- $(k+1)$  step:  $\mathbf{d}_i^{(k+1)} = \mathbf{D}_{ii}^{-1} \left[ \sum_{j \in \mathcal{N}_i, j=i} \mathbf{B}_{ij}\mathbf{d}_j^{(k)} - \mathbf{g}_i \right]$ .
11:   end for
12:   Update local iterate:  $\mathbf{x}_i = \mathbf{x}_i + \epsilon \mathbf{d}_i^{(K)}$ .
13: end while

```

---

node  $i$ . The gradient component  $\mathbf{g}_{i,t} = (1 - w_{ii})\mathbf{x}_{i,t} - \sum_{j \in \mathcal{N}_i} w_{ij}\mathbf{x}_{j,t} + \alpha \nabla f_i(\mathbf{x}_{i,t})$  is also stored and computed at  $i$ . Node  $i$  can also evaluate the values of the matrix blocks  $\mathbf{B}_{ij} = w_{ij}\mathbf{I}$ . Thus, if the NN- $k$  step components  $\mathbf{d}_{j,t}^{(k)}$  are available at neighboring nodes  $j$ , node  $i$  can then determine the NN- $(k+1)$  step component  $\mathbf{d}_{i,t}^{(k+1)}$  upon being communicated that information.

The expression in (17) represents an iterative computation embedded inside the NN- $K$  recursion in (15). For each time index  $t$ , we compute the local component of the NN-0 step  $\mathbf{d}_{i,t}^{(0)} = -\mathbf{D}_{ii,t}^{-1}\mathbf{g}_{i,t}$ . Upon exchanging this information with neighbors we use (17) to determine the NN-1 step components  $\mathbf{d}_{i,t}^{(1)}$ . These can be exchanged and plugged in (17) to compute  $\mathbf{d}_{i,t}^{(2)}$ . Repeating this procedure  $K$  times, nodes ends up having determined their NN- $K$  step component  $\mathbf{d}_{i,t}^{(K)}$ .

The NN- $K$  method is summarized in Algorithm 1. The descent iteration in (15) is implemented in Step 9. Implementation of this descent requires access to the NN- $K$  descent direction  $\mathbf{d}_{i,t}^{(K)}$  which is computed by the loop in steps 4-8. Step 4 initializes the loop by computing the NN-0 step  $\mathbf{d}_{i,t}^{(0)} = -\mathbf{D}_{ii,t}^{-1}\mathbf{g}_{i,t}$ . The core of the loop is in Step 7 which corresponds to the recursion in (17). Step 6 stands for the variable exchange that is necessary to implement Step 7. After  $K$  iterations through this loop the NN- $K$  descent direction  $\mathbf{d}_{i,t}^{(K)}$  is computed and can be used in Step 9. Both, steps 4 and 9, require access to the local gradient component  $\mathbf{g}_{i,t}$ . This is evaluated in Step 3 after receiving the prerequisite information in Step 2.

**B. Adaptive Network Newton**

As mentioned in Section II, NN- $K$  algorithm instead of solving (1) or its equivalent (7), solves a penalty version of (7) as introduced in (4). The optimal solutions of optimization problems (7) and (4) are different and the gap between them is upper bounded by  $O(\alpha)$  [8]. This observation implies that by setting a decreasing policy for  $\alpha$  or equivalently an increasing policy for penalty coefficient  $1/\alpha$ , the solution of (7) approaches the minimizer of (4), i.e.  $\tilde{\mathbf{y}}^* \rightarrow \mathbf{y}^*$  for  $\alpha \rightarrow 0$ .

We introduce Adaptive Network Newton- $K$  (ANN- $K$ ) as a version of NN- $K$  that uses a decreasing sequence of  $\alpha_t$  to achieve exact convergence to the optimal solution of (1). The idea of ANN- $K$  is to decrease parameter  $\alpha_t$  by multiplying by  $\eta < 1$ , i.e.,  $\alpha_{t+1} = \eta\alpha_t$ , when the sequence generated by NN- $K$  is converged for a specific value of  $\alpha$ . To be more precise, each node  $i$  has a signal vector  $\mathbf{s}_i = [s_{i1}; \dots; s_{in}] \in \{0, 1\}^n$  where each component is a binary variable. Note that  $s_{ij}$  corresponds to the occurrence of receiving a signal at node  $i$  from node  $j$ . Hence, nodes initialize their signaling components by 0 for all the nodes in the network. At iteration  $t$  node  $i$  computes its local gradient norm  $\|\mathbf{g}_{i,t}\|$ . If the norm of gradient is smaller than a specific value called  $\text{tol}$ , i.e.  $\|\mathbf{g}_{i,t}\| \leq \text{tol}$ , it sets the local signal component to  $s_{ii} = 1$  and sends a signal to all the nodes in the network. The receiver

**Algorithm 3** Adaptive Network Newton- $K$  method at node  $i$ 


---

**Require:** Initial iterate  $\mathbf{x}_{i,0}$ , initial penalty parameter  $\alpha_0$  and initial sequence of bits  $\mathbf{s}_i = [s_{i1}; \dots; s_{in}] = [0; \dots; 0]$ .

```

1: for  $t = 0, 1, 2, \dots$  do
2:   Call NN- $K$  function:  $\mathbf{x}_{i,t+1} = \text{NN-}K(\alpha_t, \mathbf{x}_{i,t}, \text{tol})$ 
3:   Set  $s_{ii} = 1$  and broadcast scalar it to all nodes.
4:   Set  $s_{ij} = 1$  for all nodes  $j$  that sent a signal.
5:   if  $s_{ij} = 1$  for all  $j = 1, \dots, n$  then
6:     Update penalty parameter  $\alpha_{t+1} = \eta\alpha_t$ .
7:     Set  $s_{ij} = 0$  for all  $j = 1, \dots, n$ .
8:   end if
9: end for

```

---

nodes set the corresponding component of node  $i$  in their local signal vectors to 1, i.e.  $s_{ji} = 1$  for  $j \neq i$ . This procedure implies that the signal vectors of all nodes in the network are always synchronous. The update for parameter  $\alpha_t$  occurs when all the components of signal vector are 1 which is equivalent to achieving the required accuracy for all nodes in the network. Since the number of times that  $\alpha_t$  should be updated is small, the cost of communication for updating  $\alpha_t$  is affordable.

The ANN- $K$  method is summarized in Algorithm 3. At each iteration of ANN- $K$  algorithm at Step 2 function NN- $K$  Step is called to update variable  $\mathbf{x}_{i,t}$  for node  $i$ . Note that function NN- $K$  which is introduced in Algorithm 2, runs NN- $K$  step until the time that norm of local gradient is smaller than a threshold  $\|\mathbf{g}_i\| \leq \text{tol}$ . After achieving this accuracy, in Steps 3 node  $i$  updates its local signal component  $s_{ii}$  to 1 and sends it to the other nodes. In Step 4 each node  $i$  updates the signal vector components of other nodes in the network. Then, in Step 6 the nodes update the penalty parameter for the next iteration as  $\alpha_{t+1} = \eta\alpha_t$  if all the components of signal vector is 1, otherwise they use the previous value  $\alpha_{t+1} = \alpha_t$ . In order to reset the system after updating  $\alpha_t$ , all signal vectors are set to 0, i.e.  $\mathbf{s}_i = \mathbf{0}$  for  $i = 1, \dots, n$  as in Step 7.

**III. CONVERGENCE ANALYSIS**

In this section we show that as time progresses the sequence of objective function  $F(\mathbf{y}_t)$  defined in (4) approaches the optimal objective function value  $F(\mathbf{y}^*)$  by considering the following assumptions.

**Assumption 1** *There exists constants  $0 \leq \delta < \Delta < 1$  that lower and upper bound the diagonal weights for all  $i$ ,*

$$0 \leq \delta \leq w_{ii} \leq \Delta < 1 \quad i = 1, \dots, n. \quad (18)$$

**Assumption 2** *The eigenvalues of local objective function Hessians  $\nabla^2 f_i(\mathbf{x})$  are bounded with positive constants  $0 < m \leq M < \infty$ , i.e.*

$$m\mathbf{I} \preceq \nabla^2 f_i(\mathbf{x}) \preceq M\mathbf{I}. \quad (19)$$

**Assumption 3** *The local objective function Hessians  $\nabla^2 f_i(\mathbf{x})$  are Lipschitz continuous with parameter  $L$  with respect to Euclidian norm,*

$$\|\nabla^2 f_i(\mathbf{x}) - \nabla^2 f_i(\hat{\mathbf{x}})\| \leq L \|\mathbf{x} - \hat{\mathbf{x}}\|. \quad (20)$$

Linear convergence of objective function  $F(\mathbf{y}_t)$  to the optimal objective function  $F(\mathbf{y}^*)$  is shown in [14] which we mention as a reference.

**Theorem 1** *Consider the NN- $K$  method as defined in (10)-(15) and the objective function  $F(\mathbf{y})$  as introduced in (4). If the stepsize  $\epsilon$  is chosen as  $\epsilon = \min\{1, \epsilon_0\}$  where  $\epsilon_0$  is a constant that depends on problem parameters, and Assumptions 1, 2, and 3 hold true, the sequence  $F(\mathbf{y}_t)$  converges to the optimal argument  $F(\mathbf{y}^*)$  at least linearly with constant  $0 < 1 - \zeta < 1$ . I.e.,*

$$F(\mathbf{y}_t) - F(\mathbf{y}^*) \leq (1 - \zeta)^t (F(\mathbf{y}_0) - F(\mathbf{y}^*)). \quad (21)$$

Theorem 1 shows linear convergence of sequence of objective function  $F(\mathbf{y}_t)$ . In the following section we study the performances of NN and ANN methods via different numerical experiments.

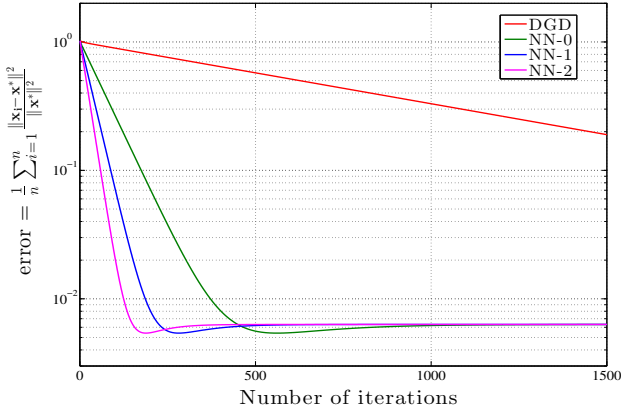


Fig. 1: Convergence of DGD, NN-0, NN-1, and NN-2 in terms of number of iterations. The NN methods converges faster than DGD. Furthermore, the larger  $K$  is, the faster NN- $K$  converges.

#### IV. NUMERICAL ANALYSIS

We compare the performance of DGD and different versions of NN in the minimization of a distributed quadratic objective. The comparison is done in terms of both, number of iterations and number of information exchanges. Specifically, for each agent  $i$  we consider a positive definite diagonal matrix  $\mathbf{A}_i \in \mathbb{S}_p^+$  and a vector  $\mathbf{b}_i \in \mathbb{R}^p$  to define the local objective function  $f_i(\mathbf{x}) := (1/2)\mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x}$ . Therefore, the global cost function  $f(\mathbf{x})$  is written as

$$f(\mathbf{x}) := \sum_{i=1}^n \frac{1}{2} \mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x}. \quad (22)$$

The difficulty of solving (22) is given by the condition number of the matrices  $\mathbf{A}_i$ . To adjust condition numbers we generate diagonal matrices  $\mathbf{A}_i$  with random diagonal elements  $a_{ii}$ . The first  $p/2$  diagonal elements  $a_{ii}$  are drawn uniformly at random from the discrete set  $\{1, 10^{-1}, \dots, 10^{-\xi}\}$  and the next  $p/2$  are uniformly and randomly chosen from the set  $\{1, 10^1, \dots, 10^\xi\}$ . This choice of coefficients yields local matrices  $\mathbf{A}_i$  with eigenvalues in the interval  $[10^{-\xi}, 10^\xi]$  and global matrices  $\sum_{i=1}^n \mathbf{A}_i$  with eigenvalues in the interval  $[n10^{-\xi}, n10^\xi]$ . The condition numbers are typically  $10^{2\xi}$  for the local functions and  $10^\xi$  for the global objectives. The linear terms  $\mathbf{b}_i^T \mathbf{x}$  are added so that the different local functions have different minima. The vectors  $\mathbf{b}_i$  are chosen uniformly at random from the box  $[0, 1]^p$ .

For the quadratic objective in (22) we can compute the optimal argument  $\mathbf{x}^*$  in closed form. We then evaluate convergence through the relative error that we define as the average normalized squared distance between local vectors  $\mathbf{x}_i$  and the optimal decision vector  $\mathbf{x}^*$ ,

$$e_t := \frac{1}{n} \sum_{i=1}^n \frac{\|\mathbf{x}_{i,t} - \mathbf{x}^*\|^2}{\|\mathbf{x}^*\|^2}. \quad (23)$$

The network connecting the nodes is a  $d$ -regular cycle where each node is connected to exactly  $d$  neighbors and  $d$  is assumed even. The graph is generated by creating a cycle and then connecting each node with the  $d/2$  nodes that are closest in each direction. The diagonal weights in the matrix  $\mathbf{W}$  are set to  $w_{ii} = 1/2 + 1/2(d+1)$  and the off diagonal weights to  $w_{ij} = 1/2(d+1)$  when  $j \in \mathcal{N}_i$ .

In the subsequent experiments we set the network size to  $n = 100$ , the dimension of the decision vectors to  $p = 4$ , the condition number parameter to  $\xi = 2$ , the penalty coefficient inverse to  $\alpha = 10^{-2}$ , and the network degree to  $d = 4$ . The NN step size is set to  $\epsilon = 1$ , which is always possible when we have quadratic objectives. Figure 1 illustrates a sample convergence path for DGD, NN-0, NN-1, and NN-2 by measuring the relative error  $e_t$  in (23) with respect to the number of iterations  $t$ . As expected for a problem that doesn't have a small condition number – in

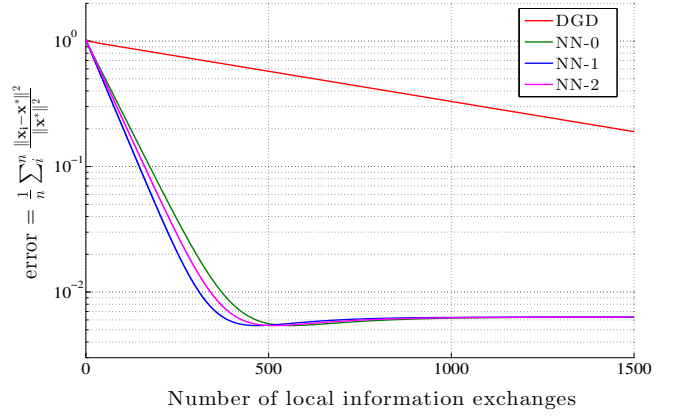


Fig. 2: Convergence of DGD, NN-0, NN-1, and NN-2 in terms of number of communication exchanges. The NN- $K$  methods retain the advantage over DGD but increasing  $K$  may not result in faster convergence. For this particular instance it is actually NN-1 that converges fastest in terms of number of communication exchanges.

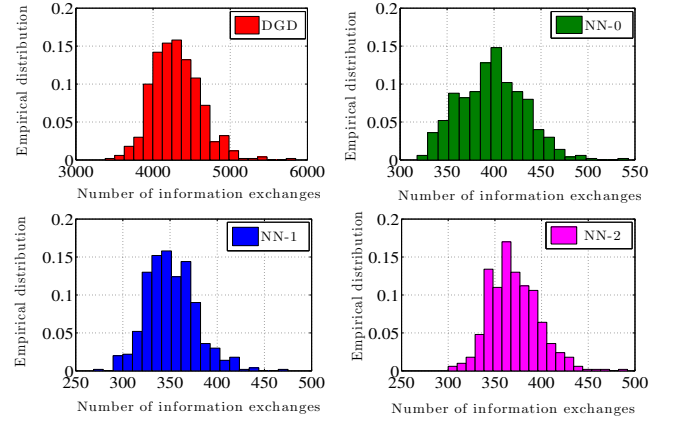


Fig. 3: Histograms of the number of information exchanges required to achieving accuracy  $e_t < 10^{-2}$ . The qualitative observations made in figures 1 and 2 hold over a range of random problem realizations.

this particular instantiation of the function in (22) the condition number is 95.2 – different versions of NN are much faster than DGD. E.g., after  $t = 1.5 \times 10^3$  iterations the error associated which DGD iterates is  $e_t \approx 1.9 \times 10^{-1}$ . Comparable or better accuracy  $e_t < 1.9 \times 10^{-1}$  is achieved in  $t = 132$ ,  $t = 63$ , and  $t = 43$  iterations for NN-0, NN-1, and NN-2, respectively.

Further recall that  $\alpha$  controls the difference between the actual optimal argument  $\tilde{\mathbf{y}}^* = [\mathbf{x}^*; \dots; \mathbf{x}^*]$  [cf. (7)] and the argument  $\mathbf{y}^*$  [cf. (4)] to which DGD and NN converge. Since we have  $\alpha = 10^{-2}$  and the difference between these two vectors is of order  $O(\alpha)$ , we expect the error in (23) to settle at  $e_t \approx 10^{-2}$ . The error actually settles at  $e_t \approx 6.3 \times 10^{-3}$  and it takes all three versions of NN less than  $t = 400$  iterations to do so. It takes DGD more than  $t = 10^4$  iterations to reach this value. This relative performance difference decreases if the problem has better conditioning but can be made arbitrarily large by increasing the condition number of the matrix  $\sum_{i=1}^n \mathbf{A}_i$ . The number of iterations required for convergence can be further decreased by considering higher order approximations in (14). The advantages would be misleading because they come at the cost of increasing the number of communications required to approximate the Newton step.

To study this latter effect we consider the relative performance of DGD and different versions of NN in terms of the number of local information exchanges. Note that each iteration in NN- $K$  requires a total of  $K + 1$  information exchanges with each neighbor, as opposed to the single

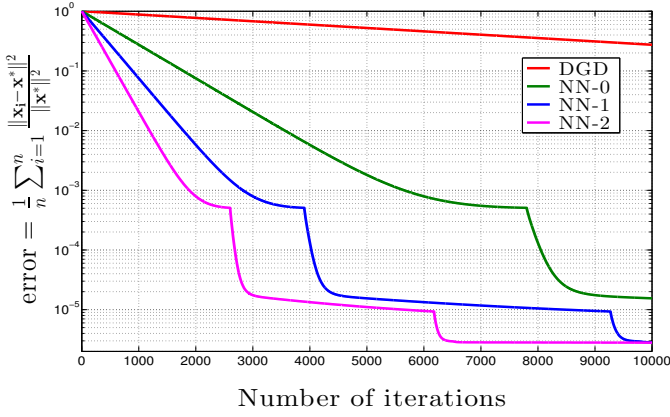


Fig. 4: Convergence of adaptive DGD, NN-0, NN-1, and NN-2 for  $\alpha_0 = 10^{-2}$ .

variable exchange required by DGD. After  $t$  iterations the number of variable exchanges between each pair of neighbors is  $t$  for DGD and  $(K+1)t$  for NN- $K$ . Thus, we can translate Figure 1 into a path in terms of number of communications by scaling the time axis by  $(K+1)$ . The result of this scaling is shown in Figure 2. The different versions of NN retain a significant, albeit smaller, advantage with respect to DGD. Error  $e_t < 10^{-2}$  is achieved by NN-0, NN-1, and NN-2 after  $(K+1)t = 3.7 \times 10^2$ ,  $(K+1)t = 3.1 \times 10^2$ , and  $(K+1)t = 3.4 \times 10^2$  variable exchanges, respectively. When measured in this metric it is no longer true that increasing  $K$  results in faster convergence. For this particular problem instance it is actually NN-1 that converges fastest in terms of number of communication exchanges.

For a more comprehensive evaluation we consider  $10^3$  different random realizations of (22) where we also randomize the degree  $d$  of the  $d$ -regular graph that we choose from the even numbers in the set  $[2, 10]$ . The remaining parameters are the same used to generate figures 1 and 2. For each joint random realization of network and objective we run DGD, NN-0, NN-1, and NN-2, until achieving error  $e_t < 10^{-2}$  and record the number of communication exchanges that have elapsed – which amount to simply  $t$  for DGD and  $(K+1)t$  for NN. The resulting histograms are shown in Figure 3. The mean times required to reduce the error to  $e_t < 10^{-2}$  are  $4.3 \times 10^3$  for DGD and  $4.0 \times 10^2$ ,  $3.5 \times 10^2$ , and  $3.7 \times 10^2$  for NN-0, NN-1, and NN-2. As in the particular case shown in figures 1 and 2, NN-1 performs best in terms of communication exchanges. Observe, however, that the number of communication exchanges required by NN-2 is not much larger and that NN-2 requires less computational effort than NN-1 because the number of iterations  $t$  is smaller.

#### A. Adaptive Network Newton

Given that DGD and NN are penalty methods it is of interest to consider their behavior when the inverse penalty parameter  $\alpha$  is decreased recursively. The adaptation of  $\alpha$  for NN- $K$  is discussed in Section II-B where it is termed adaptive (A)NN- $K$ . The same adaptation strategy is considered here for DGD. The parameter  $\alpha$  is kept constant until the local gradient components  $\mathbf{g}_{i,t}$  become smaller than a given tolerance  $\text{tol}$ , i.e., until  $\|\mathbf{g}_{i,t}\| \leq \text{tol}$  for all  $i$ . When this tolerance is achieved, the parameter  $\alpha$  is scaled by a factor  $\eta < 1$ , i.e.,  $\alpha$  is decreased from its current value to  $\eta\alpha$ . This requires the use of a signaling method like the one summarized in Algorithm 3 for ANN- $K$ .

We consider the objective in (22) and nodes connected by a  $d$ -regular cycle. We use the same parameters used to generate figures 1 and 2. The adaptive gradient tolerance is set to  $\text{tol} = 10^{-3}$  and the scaling parameter to  $\eta = 0.1$ . We consider two different scenarios where the initial penalty parameters are  $\alpha = \alpha_0 = 10^{-1}$  and  $\alpha = \alpha_0 = 10^{-2}$ . The respective error trajectories  $e_t$  with respect to the number of iterations are shown in figures 4 – where  $\alpha_0 = 10^{-2}$  – and 5 – where  $\alpha_0 = 10^{-1}$ . In each figure we show  $e_t$  for adaptive DGD, ANN-0, ANN-1, and ANN-2. Both

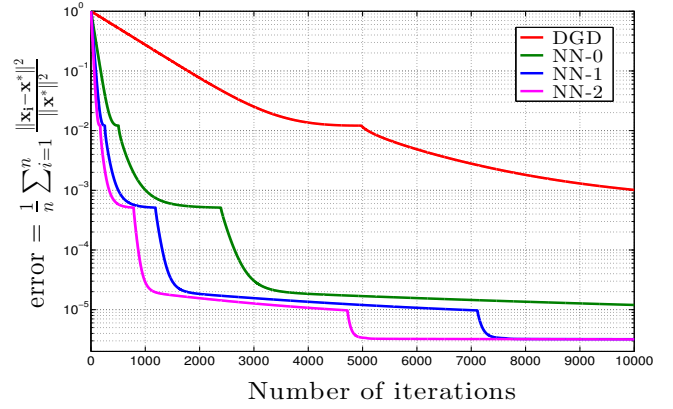


Fig. 5: Convergence of Adaptive DGD, NN-0, NN-1, and NN-2 for  $\alpha_0 = 10^{-1}$ .

figures show that the ANN methods outperform adaptive DGD and that larger  $K$  reduces the number of iterations that it takes ANN- $K$  to achieve a target error. These results are consistent with the findings summarized in figures 1-3.

More interesting conclusions follow from a comparison across figures 1 and 2. We can see that it is better to start with the (larger) value  $\alpha = 10^{-1}$  even if the method initially converges to a point farther from the actually optimum. This happens because problems with larger  $\alpha$  are better conditioned and thus easier to minimize.

#### REFERENCES

- [1] A. Ribeiro, “Ergodic stochastic optimization algorithms for wireless communication and networking,” *IEEE Trans. Signal Process.*, vol. 58, no. 12, pp. 6369–6386, December 2010.
- [2] —, “Optimal resource allocation in wireless communication and networking,” *EURASIP J. Wireless commun.*, vol. 2012, no. 272, pp. 3727–3741, August 2012, puta carajo.
- [3] I. Schizas, A. Ribeiro, and G. Giannakis, “Consensus in ad hoc wsns with noisy links - part i: Distributed estimation of deterministic signals,” *IEEE Transactions on Signal Processing*, vol. 56, pp. 350–364, 2008.
- [4] M. Rabbat and R. Nowak, “Distributed optimization in sensor networks,” *proceedings of the 3rd international symposium on Information processing in sensor networks*, pp. 20–27, ACM, 2004.
- [5] V. Cevher, S. Becker, and M. Schmidt, “Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics,” *IEEE Signal Processing Magazine*, vol. 31, pp. 32–43, 2014.
- [6] A. Nedic and A. Ozdaglar, “Distributed subgradient methods for multiagent optimization,” *IEEE Transactions on Automatic Control*, vol. 54, pp. 48–61, 2009.
- [7] D. Jakovetic, J. Xavier, and J. Moura, “Fast distributed gradient methods,” *IEEE Transactions on Automatic Control*, vol. 59, pp. 1131–1146, 2014.
- [8] K. Yuan, Q. Ling, and W. Yin, “On the convergence of decentralized gradient descent,” *arXiv preprint arXiv: 1310.7063*, 2013.
- [9] W. Shi, Q. Ling, G. Wu, and W. Yin, “Extra: An exact first-order algorithm for decentralized consensus optimization,” *arXiv preprint arXiv: 1404.6264*, 2014.
- [10] Q. Ling and A. Ribeiro, “Decentralized linearized alternating direction method of multipliers,” *Proc. Int. Conf. Acoustics Speech Signal Process.*, pp. 5447–5451, 2014.
- [11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
- [12] W. Shi, Q. Ling, G. Wu, and W. Yin, “On the linear convergence of the admm in decentralized consensus optimization,” *IEEE Transactions on Signal Processing*, vol. 62, pp. 1750–1761, 2014.
- [13] J. Duchi, A. Agarwal, and M. Wainwright, “Dual averaging for distributed optimization: Convergence analysis and network scaling,” *IEEE Transactions on Automatic Control*, vol. 57, pp. 592–606, 2012.
- [14] A. Mokhtari, Q. Ling, and A. Ribeiro, “An approximate newton method for distributed optimization,” 2014, available at <http://www.seas.upenn.edu/~aryanm/wiki/NN-ICASSP.pdf>.